Seamless Access to Adaptive Solver Algorithms

Dorian C. Arnold* Susan Blackford* Jack Dongarra* Victor Eijkhout*
Tinghua Xu*

Abstract

The NetSolve project was established to aid scientists who prefer not to be concerned with the usual tedium associated with finding and maintaining software libraries with which to create programs, toolkits and problem solving environments particular to their scientific domain. Through NetSolve, users are given access to complex algorithms that solve a variety of types of problems, one instance being linear systems solvers. All solvers, however, are not built alike; depending on the characteristics of the matrix being solved some perform poorly and others not at all. This article introduces the reader to the NetSolve system and discusses how it can be used to access a large suite of numerical solvers, and to analyse the input matrix to determine, transparent to the user, a solver and use it to find a solution to the system. We give some background on the design of the netsolve system, the interface to solver packages, and we discuss the heuristics used in transparently determining the specific solver.

Key words: Distributed Computing, Problem Solving Environments, Sparse/Dense Solvers, Linear Systems Solvers.

1 Introduction

The performance of computer processors continues to increase in accordance with, and even surpassing, Moore's Law. Computer infrastructures (i.e., buses, networks, etc.) are also becoming faster, larger and able to support more rigorous computational challenges. These benefits do not, however, come without a cost. The systems to be solved are complex and present quite a challenge to administrators, programmers, and end-users who seek to use the available horsepower in their solution. Also mainly due to non-standardization of interfaces and protocols, the same variety of systems and component choices that provide a flexibility and ability to customize computers to one's liking often proves to be an obstacle when moving from one platform to another.

In numerical computing, researchers have both developed efficient algorithms, and implemented these, so that end-users can use a package without being concerned about either the theory or the low-level details of their computer system. Users are left with two tasks: the first to decide which package is the best fit to their application, and the second to learn the peculiarities of the software package that is ultimately chosen.

The goal of the Netsolve projects is, first of all, to provide a seamless, uniform, and easy-to-use, interface to a variety of scientific libraries, including linear equation solvers, and, secondly, to implement heuristics that will choose a specific solver based on characteristics of the input coefficient matrix. To be explicit, we stress that it is not the solver algorithm itself, but rather the NetSolve system, that adapts to the characteristics of the system being solved and chooses the best algorithm for it. This work is not aimed at the expert user who wishes to effectively utilize every cycle in his application and will ultimately invest the time to make the appropriate decisions himself. We offer this work for those who wish not to be distracted by the details of the problem at hand, but are satisfied with a reasonably efficient system that automatically considers the details for them.

Section 2 of this paper presents details about NetSolve, which is our deployment environment. Section 3 discusses NetSolve's interactions with numerous linear solvers and describes the algorithms and criteria we use to determine when to use a particular solver. Finally, Section 4 summarizes the work and discusses future research goals.

*Computer Science Department, University of Tennessee, Knoxville, TN 37996. [darnold, susan, dongarra, eijkhout, xu]@cs.utk.edu

16th IMACS World Congress (© 2000 IMACS)

1.1 Related Work

There is little existing work on automatically determining the best solver for a given system. For a comparison study of solvers and preconditioners, we refer to [14].

2 Network-enabled Solvers

The NetSolve project is being developed at the University of Tennessee. Its original motivation was to alleviate the difficulties that domain scientists usually encounter when trying to locate, install, and use numerical software, especially on multiple platforms. NetSolve provides remote access to computational resources, both hardware and software. Built upon standard Internet protocols, such as TCP/IP sockets, it is available for all popular variants of the UNIX operating system, and parts of the system are available for the Microsoft Windows 95, 98 and NT platforms.

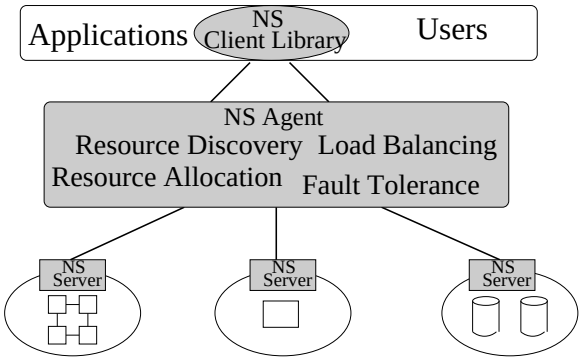

Figure 1: Architectural Overview of the NetSolve System

Figure 1 shows the infrastructure of the NetSolve system and its relation to the applications that use it. NetSolve and systems like it are often referred to as Grid middleware; this figure helps to clarify this choice in terminology. The shaded parts of the figure represent the NetSolve system. It can be seen that NetSolve acts as a glue layer that brings the application or user together with the hardware and/or software needed.

At the top tier, the NetSolve client library is linked in with the user's application. The application then makes calls to NetSolve's application programming interface (API) for specific services. Through the API, NetSolve client-users gain access to aggregate resources without needing to know anything about computer networking or distributed computing. In fact, the user does not even have to know which remote resources are involved.

```
...                                  ...
A = read_matrix();                   A = read_matrix();
B = read_matrix();                   B = read_matrix();
C = matmul(A, B);                    status = netsolve("matmul", A, B, C);
....                                 ...
```

Figure 2: Sample C code: Left side before NetSolve, right side after NetSolve integration

Figure 2 illustrates what the programming code would look like before and after the NetSolve API has been integrated. The (hidden) semantics of a NetSolve request are:

1. Client contacts the agent for a list of capable servers.
2. Client contacts server and sends input parameters.
3. Server runs appropriate service.
4. Server returns output parameters or error status to client.

There are many advantages to using a system like NetSolve. NetSolve provides access to otherwise unavailable software. In cases where the software is at hand, it can make the power of supercomputers accessible from low-end machines such as notebook computers. Furthermore, as explained below, NetSolve adds heuristics that attempt to find the most expeditious route to solve any given problem. NetSolve currently supports the C, FORTRAN, Matlab, and Mathematica programming interfaces as languages of implementation for client programs.

The NetSolve agent represents the gateway to the NetSolve system. It maintains a database of NetSolve servers along with their capabilities (hardware performance and allocated software) and dynamic usage statistics. It uses this information to allocate server resources for client requests. The agent, in its resource allocation mechanism, attempts to find the server that will service the request the quickest, balance the load amongst its servers and keep track of failed servers. Requests are directed away from failed servers. The agent also adds fault-tolerance features that attempt to use every likely server until it finds one that successfully services the request.

The NetSolve server is the computational backbone of the system. It is a daemon process that awaits client requests. The server can run on single workstations, clusters of workstations, symmetric multi-processors or machines with massively parallel processors. A key component of the NetSolve server is a source code generator which parses a NetSolve problem description file (PDF). This PDF contains information that allows the NetSolve system to create new modules and incorporate new functionalities. In essence, the PDF defines a wrapper that NetSolve uses to call the function being incorporated.

For more detailed information on the NetSolve system and its usage, refer to [3] and [4]. The next major release of NetSolve is planned for April of 2000. Features to be implemented in this release include a Java GUI to aid in the creation of PDFs, a Microsoft Excel interface, more object datatypes, more server modules included with the distribution, and enhanced load balancing. Currently, NetSolve-1.2 and a beta version of NetSolve-1.3, including APIs for the Win32 platform, can be downloaded from the project web site at **www.cs.utk.edu/netsolve**. A version implementing the work described here will be available around September, 2000. NetSolve has been recognized as a significant effort in research and development, and was named in R&D Magazine's top 100 list for 1999.

3 NetSolve and Linear Solvers

In this section we provide some details regarding the interface to equation solver libraries, and we discuss the implementation of the 'equation solver expert agent'. An important point is that for the latter we do not change the syntax of the existing NetSolve interface, instead defining a new problem, **LinearSolve**, that instucts NetSolve to proceed with analysis of the input matrix.

Using a NetSolve interface to a numerical library alleviates the user's need to learn the intricacies of a specific software package (or its auxiliaries, such as MPI or BLAS). After investing the time to learn the data structures and specifics of a particular library, the user may then determine that there are important classes of linear systems for which the package may not be well-suited. At this point, the only choice is to repeat (possibly several times) the procedure of learning a new software package. It is only now that he can create his application that uses these solvers, which was the only thing that interested him in the first place.

NetSolve provides an easy interface to a variety of linear systems solvers. We have done the tedious work of learning (and even developing) a variety of systems solver interfaces. The point of interest to the end-user is that via NetSolve, he can seamlessly access all these solvers in a uniform way. We have integrated numerous solvers, including some from LAPACK [1], ARPACK [8], PETSc [2], Aztec [7], SuperLU [9] and MA28 [5]. Each of these packages has its own set of options and settings. We have used the PDF, as described in Section 2 to make these functions available from the NetSolve interface. After installing any of the NetSolve client interfaces, the user can then access our pool of NetSolve servers to not only access the routines, but also computational cycles and other resources that have been devoted to the system.

Examples 3.1 `[x, its] = netsolve('petsc', A, b, 1.e-6, 500);`

Example 3.1 invokes PETSc from the Matlab client interface. Here, `A` is the sparse coefficient matrix, `b` is the right hand side, `1.e-6` is the error tolerance and `500` is the maximum number of iterations. In the output set, `x` is the solution vector, and `its` is the number of iterations that was achieved. By simply changing the string `petsc` to one of the other available options, say `aztec`, the user can change which library will be invoked to solve the problem.

If `A` is a dense coefficient matrix, one could call the *LU* driver routine from LAPACK as follows:

Examples 3.2 `[lu, p, x, info] = netsolve('dgesv', A, b);`

where `lu` is the LU factors (A = P*L*U), `p` is the vector of pivots, `info` indicates success, failure, or whether the LU algorithm failed, and all the other parameters are the same as in the PETSc example.

3.1 NetSolve "LinearSolve" Interface

The NetSolve "LinearSolve" interface makes the decision of which library and routine to choose for a given problem, and is provided for a user who does not want the burden of choosing a specific library, or perhaps does not know which library is most appropriate for his application. This interface makes the decisions for the user.

Examples 3.3 `[x] = netsolve('LinearSolve', A, b);`

From Matlab, the function call in Example 3.3 will use the heuristics discussed below to determine which package to use. The user is guaranteed to get back the proper results if a server is available, and trusts NetSolve to yield that result using what it sees as the best algorithm. When a user requests a linear system to be solved using this blackbox interface, a number of decisions must be made based on the structure of the matrix `A`, to determine the exact solver to be chosen.

3.1.1 Matrix shape

We allow the user to submit both rectangular and square matrices to the blackbox solver. For rectangular matrices we use a linear least squares solver from LAPACK.

3.1.2 Matrix element density

For square systems, there is a distinction between sparse and dense matrices. The NetSolve system natively supports data objects, two of which are Matrices (regular, dense) and Sparse Matrices. However, since within the NetSolve framework, problems must specify the exact type of input object, the `LinearSolve` routine takes a regular dense matrix as input. We manually check the percentage of nonzeros, and transform the object to a sparse matrix if the nonzeroes do not number more than a certain threshhold. For dense systems, the only further property we determine is their symmetry. We solve these systems using LAPACK. There are dense matrices, for instance from boundary elements, that can be solved by iterative methods more efficiently than by direct methods, but there is no blackbox way of recognising such systems.

In the sparse case, we try to discover several special cases:

- Banded matrices with a relatively dense band can be solved by a direct method from LAPACK.
- We try to discover a block or diagonal structure in the matrix for such packages as support this; for instance, Aztec. Using block methods can lead to a higher performance than for general sparse methods, since they enable use of Blas3-based algorithms.

In the sparse matrix case, there is a further special case that we try to catch, namely that of matrices with a zero $(2, 2)$ block. These are often indefinite matrices coming from either optimization or mixed finite elements, and there are special algorithms for these.

3.1.3 Sparse Solvers

The main question for sparse matrices is whether to use a direct or an iterative solver. First of all we observe that iterative methods work well mostly on matrices that come from elliptic partial differential equations. Thus, they are unlikely to perform well on, for instance, the matrices coming from large systems of ordinary differential equations. We use some simple heuristics to find such systems, for instance by testing for a diagonal that is mostly zero.

For the remaining matrices there are two factors on which to base a choice between a direct and an iterative method.

size of the matrix For big matrices, the fill-in of a direct method can be much larger than the memory demands of an iterative method, even with a complicated preconditioner.

numerical properties In general, direct solvers can handle much more complicated matrices than iterative methods; one can say that iterative methods work best if the matrix is not too far from definite.

Thus we are left with the questions how to estimate the amount of fill-in and how to gauge numerical properties.

3.1.4 Fill-in estimation

The argument against direct methods is usually the storage demand. This, however, is quite reasonable if the matrix graph stems from a two-dimensional PDE, in which case a generalised nested dissection [10, 11] factorisation needs relatively little storage, typically $O(N \log N)$. Thus, even a fairly large sparse matrix might be amenable to solution by a direct method. Fortunately, the determination of the fill-in of a direct method can be performed in linear time. We then make this determination in order to see whether the available servers have enough memory to accomodate a direct solver.

3.1.5 Issues in iterative solvers

For a blackbox iterative solver, we need to supply automatically such parameters as the number of iterations. If the method reaches the indicated maximum, it can be that the method is not making any progress, or it can be that there was progress but a larger number of iterations is needed. If an iterative solver package lets us monitor the convergence history, we can make an estimate of the number of iterations required, and we restart the method. Otherwise, we may switch to a direct solver. In the current implementation we use a BiCGstab solver [13] with Jacobi preconditioning, but we will extend the intelligence of the agent in making a more sophisticated determination.

4 Conclusion and Future Work

In this paper, we have presented the techniques and implementation of a system, namely NetSolve, that can be used to transparently decide upon suitable linear solver packages and use them over distributed resources to solve different classes of systems of equations. Our approach is to use NetSolve's existing components to add the capabilities of different linear solver libraries to the system and then add heuristics to analyse input sets and determine which package should be used. While this analysis adds overhead, its benefit is that it alleviates the user of the tedium of making this choice for himself.

Our plans are to further refine our techniques in order to reap more benefits from our system. Continuing to investigate the conditions under which different solver toolkits perform best is one way in which we plan to do this. We also plan to use our framework to gather empirical data that support and validate our theories. An idea we have to aid us in determining the spectral properties of a matrix when using iterative methods is to run a few iterations of the conjugate gradient method to see if the system breaks down. If no breakdown occurs, the matrix is likely well-behaved; otherwise, we can attempt to make the matrix diagonally dominant in order to see if a method such as QMR [6] or GMRES [12] might work. We also plan to extend this algorithm selection technique to encompass eigensolvers and other types of solvers.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] S. Balay, W. D. Gropp, and B. F. Smith. *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[3] H. Casanova and J. Dongarra. NetSolve's Network Enabled Server: Examples and Applications. *IEEE Computational Science & Engineering*, 5(3):57–67, September 1998.

[4] H. Casanova, J. Dongarra, and K. Seymour. Client User's Guide to Netsolve. Technical Report CS-96-343, Department of Computer Science, University of Tennessee, 1996.

[5] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matices*. Clarendon Press, Oxford, 1986.

[6] R. W. Freund and N. M. Nachtigal. QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems. *Numerische Mathematik*, 60:315–339, 1991.

[7] S. A Hutchinson, Shadid J. N., and Tuminaro R. S. Aztec user's guide: Version 1.1. Technical Report SAND95-1559, Sandia National Laboratories, 1995.

[8] R. Lehoucq, D. Sorensen, and C. Yang. *ARPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, first edition, 1998.

[9] X. Li. *Sparse Gaussian Elimination on High Performance Computers*. PhD thesis, University of California at Berkeley, 1996. Computer Science Dept.

[10] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized Nested Dissection. *SIAM Journal on Numerical Analysis*, 16:346–358, 1979.

[11] R. J. Lipton and R. E. Tarjan. A Separator Theorem for Planar Graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[12] Y. Saad and M. H. Schultz. GMRes: A Generalized Minimal Residual Algorithm for Solving non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[13] Henk van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[14] Ulrike Meier Yang. Preconditioned conjugate gradient-like methods for nonsymmetric linear systems. Technical Report 1210, Center for Supercomputing Research and Development (CSRD), University of Illinois Urbana-Champaign.